1   We claim:

2   1.   A method for implementing two types of architectures on a chip, comprising:

3   receiving an instruction from a fetch engine,

4   determining whether the instruction is a macroinstruction or a microinstruction,

5   if the instruction is a macroinstruction,

6   sending the macroinstruction to an emulation engine,

7   decomposing the macroinstruction into one or more microinstructions,

8   formatting, by a bundler, the microinstructions into bundles as preferred by

9   the native microarchitecture,

10   dispatching a bundle in parallel to an execution engine,

11   if the instruction is microinstruction, dispatching the microinstruction to the

12   execution engine; and

13   dispatching additional information to the execution engine, wherein the

14   additional information is contained in bits of the bundle otherwise not

15   required for emulation of the macroinstruction.

16   2.   The method of claim 1, further comprising,

17   selecting either the microinstruction from the fetch engine or the bundle from the

18   emulation engine, by using a multiplexer, and

19   dispatching the selected instruction to the execution engine.

20   3.   The method according to clam 1 wherein the additional information includes

21   control information from the emulation front end that is sent using a memory, floating-

22   point, integer ("MFI") template, wherein the MFI template specifies that the bundle

23   includes a memory instruction in a first syllable, a floating point instruction in a second

24   syllable, and an integer instruction in a third syllable.

25   4.   The method according to clam 1 wherein the additional information includes an

26   immediate from an emulation front end that is sent by using a memory, long-immediate,

27   integer ("MLI") template that is interpreted by the execution engine differently,

28   depending upon whether the execution engine is operating in native mode or emulation

29   mode.

30   5.   The method according to clam 1, wherein the bundler

31   receives at least one sequence of instructions ("XUOPs"),

32   determines how many XUOPs are received, and

33   when more than one XUOP is received, determines whether the XUOPs

34   must be issued in parallel.

1 6.   The method of claim 5, wherein the step of dispatching comprising dispatching a

2 plurality of the bundles containing XUOPs when a plurality of XUOPs must be issued in

3 parallel, and

4   the step of issuing the XUOPs in parallel uses the following rules:

5   when the XUOPs must be issued in parallel, issues a plurality of

6 the bundles containing the XUOPs to the execution engine in parallel,

7   when the XUOPs need not be issued in parallel, determines

8 whether a particular problem exists, and

9   when the problem does not exist, dispatches a plurality of

10 the bundles containing the XUOPs to the execution engine in parallel,

11   when the problem does exist, dispatches a plurality of the

12 bundles containing one of the XUOPs to the execution engine,

13   when only one XUOP is received, determines whether the one XUOP must

14 be issued in parallel with another XUOP, and

15   when the one XUOP must be issued in parallel, dispatches nothing

16 to the execution engine,

17   when the one XUOP need not be issued in parallel, dispatches the

18 bundle containing the one XUOP to the execution engine.

19 7.   A method for implementing two architectures on a chip, comprising,

20   decoding a macroinstruction into one or more microinstructions, through the use

21 of an emulation engine,

22   formatting the microinstructions into bundles, by use of a bundler, as preferred by

23 the native microarchitecture, wherein the bundler

24   receives at least one sequence of instructions (an "XUOP"),

25   determines how many of the at least one XUOP are received, and

26   when more than one XUOP is received, determines whether the XUOPs

27 must be issued in parallel,

28   dispatching the bundle to an execution engine, and

29   dispatching additional information to the execution engine, wherein the

30 additional information is contained in bits of the bundle otherwise not

31 required for emulation of the macroinstruction.

32 8.   The method according to clam 7 wherein the additional information includes an

33 immediate from an emulation front end that is sent by using a memory, long-immediate,

34 integer ("MLI") template that is interpreted by the execution engine differently,

1 depending upon whether the execution engine is operating in native mode or emulation
2 mode.

3 9. The method of claim 8, wherein, when the execution engine is operating in native
4 mode, the MLI template specifies that a third syllable of the bundle contains an integer
5 instruction that operates on an immediate located in second and third syllables of the
6 bundle, and, when the execution engine is operating in emulation mode, the MLI template
7 specifies that the third syllable of the bundle contains an integer instruction that operates
8 on an immediate located entirely within the second syllable.

9 10. The method according to clam 7 wherein the additional information includes
10 control information from the emulation front end that is sent using a memory, floating-
11 point, integer ("MFI") template, wherein the MFI template specifies that the bundle
12 includes a memory instruction in a first syllable, a floating point instruction in a second
13 syllable, and an integer instruction in a third syllable.

14 11. The method of claim 7, wherein the emulation engine delivers a pre-decode bit to
15 the execution engine along with the bundle.

16 12. The method of claim 7, wherein the step of determining whether the XUOPs must
17 be issued in parallel uses the following rules:

18 when the XUOPs must be issued in parallel, issues a plurality of
19 the bundles containing the XUOPs to the execution engine in parallel,

20 when the XUOPs need not be issued in parallel, determines
21 whether a particular problem exists, and

22 when the problem does not exist, dispatches a plurality of
23 the bundles containing the XUOPs to the execution engine in parallel,

24 when the problem does exist, dispatches a plurality of the
25 bundles containing one of the XUOPs to the execution engine,

26 when only one XUOP is received, determines whether the one XUOP must
27 be issued in parallel with another XUOP, and

28 when the one XUOP must be issued in parallel, dispatches nothing
29 to the execution engine,

30 when the one XUOP need not be issued in parallel, dispatches the
31 bundle containing the one XUOP to the execution engine.

32 13. A method for implementing two architectures on a chip, comprising:

33 decoding a macroinstruction into one or more microinstructions, through the use
34 of an emulation engine,

1   converting the one or more microinstructions into a bundle, using a bundler, the

2   bundle having at least one syllable and having a template that specifies a type of data

3   included in the bundle, wherein the emulation engine delivers a pre-decode bit to the

4   execution engine along with the bundle, and wherein the bundler

5   receives at least one sequence of instructions (an "XUOP"),

6   determines how many of the at least one XUOP are received, and

7   when more than one XUOP is received,

8   determines whether the XUOPs must be issued in parallel, and

9   when the XUOPs must be issued in parallel, issues a plurality of

10   the bundles containing the XUOPs to the execution engine in parallel,

11   when the XUOPs need not be issued in parallel, determines

12   whether a particular problem exists, and

13   when the problem does not exist, dispatches a plurality of

14   the bundles containing the XUOPs to the execution engine in parallel,

15   when the problem does exist, dispatches a plurality of the

16   bundles containing one of the XUOPs to the execution engine,

17   when only one XUOP is received, determines whether the one XUOP must

18   be issued in parallel with another XUOP, and

19   when the one XUOP must be issued in parallel, dispatches nothing

20   to the execution engine,

21   when the one XUOP need not be issued in parallel, dispatches the

22   bundle containing the one XUOP to the execution engine,

23   dispatching the bundle to an execution engine together with a pre-decode bit, and

24   transferring, by the emulation engine, additional information to the execution

25   engine, wherein the additional information includes an immediate from an emulation

26   front end that is sent by using an memory, long-immediate, integer ("MLI") template that

27   is interpreted by the execution engine differently, depending upon whether the execution

28   engine is operating in native mode or emulation mode.

29   14.   The method of claim 13, wherein the additional information is contained in bits of

30   the bundle otherwise not required for emulation of the macroinstruction.

31   15.   The method of claim 13, wherein, when the execution engine is operating in

32   native mode, the MLI template specifies that a third syllable of the bundle contains an

33   integer instruction that operates on an immediate located in second and third syllables of

34   the bundle, and, when the execution engine is operating in emulation mode, the MLI

1 template specifies that the third syllable of the bundle contains an integer instruction that

2 operates on an immediate located entirely within the second syllable.

3 16. A method for implementing two architectures on a chip, comprising:

4 decoding a macroinstruction into one or more microinstructions, through the use

5 of an emulation engine,

6 converting the one or more microinstructions into a bundle, using a bundler, the

7 bundle having at least one syllable and having a template that specifies a type of data

8 included in the bundle, wherein the emulation engine delivers a pre-decode bit to the

9 execution engine along with the bundle, and wherein the bundler

10 receives at least one sequence of instructions (an "XUOP"),

11 determines how many of the at least one XUOP are received, and

12 when more than one XUOP is received,

13 determines whether the XUOPs must be issued in parallel, and

14 when the XUOPs must be issued in parallel, issues a plurality of

15 the bundles containing the XUOPs to the execution engine in parallel,

16 when the XUOPs need not be issued in parallel, determines

17 whether a particular problem exists, and

18 when the problem does not exist, dispatches a plurality of

19 the bundles containing the XUOPs to the execution engine in parallel,

20 when the problem does exist, dispatches a plurality of the

21 bundles containing one of the XUOPs to the execution engine,

22 when only one XUOP is received, determines whether the one XUOP must

23 be issued in parallel with another XUOP, and

24 when the one XUOP must be issued in parallel, dispatches nothing

25 to the execution engine,

26 when the one XUOP need not be issued in parallel, dispatches the

27 bundle containing the one XUOP to the execution engine,

28 dispatching the bundle to an execution engine together with a pre-decode bit, and

29 transferring, by the emulation engine, additional information to the execution

30 engine, wherein the additional information including control information from the

31 emulation front end that is sent using a memory, floating-point, integer ("MFI") template,

32 wherein the MFI template specifies that the bundle includes a memory instruction in a

33 first syllable, a floating point instruction in a second syllable, and an integer instruction in

34 a third syllable.

1   17.   The method of claim 16, wherein the additional information is contained in bits of

2   the bundle otherwise not required for emulation of the macroinstruction.